SYSTEMS DEVELOPMENT AND PROGRAMMING

CHAPTER 12

(FILE NAME ON DISK # 1 = S2C12.WPD)

All information systems processing functions within a systems development or maintenance process should be guided by written standards and procedures. The standards should address systems design, programming, testing, systems implementation, documentation, and software maintenance. They should include guidelines for effective project controls and procedures for reviewing and acquiring software systems. Standards provide a means of determining if the installation is meeting established policy. By defining uniform methods for performing routine tasks, the standards manual can affect profits and efficiency favorably. When used as a training tool, it enables new employees to become productive in a shorter period of time. Systems and programming standards must identify control procedures to ensure program integrity, restrict unauthorized access, and provide adequate systems documentation. The procedures should identify physical restrictions, software controls, and accounting controls required to maintain security within the application systems, operating systems, and data files. The content and format of the standards and procedures manual will vary with the size of the installation. Larger installations may have a multi-volume manual, where each volume addresses a single topic, such as system design, programming, documentation, and operations. Smaller data centers may cover these areas in a single manual. All installations should establish basic written operating standards. The coordination of these standards into an orderly document improves overall organization and control.

Documentation is a record of procedures for performing information systems processing tasks. It is an accounting of both the essential elements of the information systems application and the logic of the computer software programs. Documentation is the basic source of information for those who audit, correct, improve, manage, operate, or use the system. Program documentation is one of the most vital and neglected areas of information systems. It requires input from persons who have insight, the ability to think logically, and an adequate knowledge of programming languages. Management must participate actively in developing appropriate standards for documentation. To a large extent, the achievement of documentation objectives is directly related to the willingness of senior management to establish the necessary standards.

The relatively recent concept of computer aided software engineering (CASE) automates the process of developing and maintaining software.

PROJECT CONTROL

A key factor in a well-managed information systems facility is the process used to control projects for systems and applications development, acquisition, or revision. Often the difference between a successful project and a problem one is the effectiveness of project control. Project management, however, is not a substitute for technical skills.

Systems development must be monitored closely to control costs and ensure the creation of well structured applications. In a relatively small facility, a formal project control system may not be required. If management frequently meets with systems development personnel and uses adequate management techniques, it can effectively control projects. However, a more formal approach to project control is often preferred, and necessary, in many facilities.

A project control system should employ well-defined and proven techniques for managing projects and generating application development records. Each development task should be identified and tracked to determine its status. Tasks are commonly grouped into major development phases to identify significant project milestones. Although formal project control systems differ significantly, they should at a minimum contain:

- Target completion dates for each task or phase of systems development. A final project completion date is determined by carefully assessing all tasks to be performed. Identification of target dates for tasks or phases provides the substance of project control. The status of the project can be determined by comparing actual completion dates for parts of the project against planned targets. Project status reports should be used to present such comparisons for managerial review.

- Measurement of progress against original target dates. Project monitoring loses significance when original target dates are revised continually. Although it may be necessary to revise target dates, progress should be measured against original targets to better assess time and cost overruns. If development cost overruns become substantial, the justification for the project may need to be reexamined.

- Project control data aids in managing the system and programming function. Such data is appropriate for periodic reports to senior management on project priorities, project status, resource allocations, target deviations, and budgets. Examples of such project planning tools include Critical Path Method (CPM), (PERT) Program Evaluation and Review Technique, and Gantt charts.

SYSTEMS DEVELOPMENT STANDARDS

Development of computer application systems involves many activities. Standards and procedures should govern each activity. Standards should be written for:

- *System design* – To guide the creation of effective, efficient, and control-oriented application systems.

- *Software analysis and selection* – To ensure the acquisition of quality software that meets the institution's need.

- *Programming* – To ensure that desired features and controls are included in application programs.

- *Testing* – To confirm that all programs and systems are tested adequately. Tests should encompass all conceivable data quality or processing problems.

- *Implementation* – To ensure that a newly developed system is complete, that it functions as specified, and that personnel responsible for its operation (user and computer operations) are adequately trained.

- *Cataloging* – To control additions and changes to the production program library to ensure their integrity.

- *Modifications* – To guard against unauthorized software changes.

- *Documentation* – To assess the effectiveness of written material supporting the systems developed or programs changed.

These standards can be used to guide and monitor the activities of systems analysts and application programmers.

Systems Development Life Cycle

A common approach to the management of application software is the process known as a Systems Development Life Cycle (SDLC) also some times called Waterfall System Development. There are several significant control issues regarding the use of traditional SDLC methods with large-scale integrated systems. Current system development techniques may not permit the timely development and implementation of a complex system. SDLC techniques may need to be revamped to provide more increased flexibility. However, control and management methods may vary according to the complexity of the system under development.

Minimum SDLC standards should ensure that project development is sufficiently controlled to ensure the integrity of the system. Testing of various stages within large scale integrated systems may require innovative techniques. Reference SP-4.

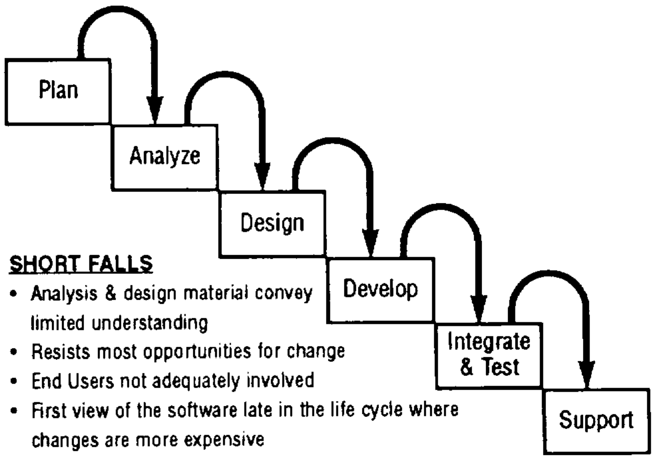
Management should carefully consider the cost of the extensive user involvement in the system development stage. User involvement is necessary to ensure the successful implementation of a large scale integrated system.

Nevertheless, management must provide more comprehensive employee training since the adoption of a LSIS will affect all departments.

SDLC standards need to be flexible, while still providing maintenance of system integrity during development to maintain that a system of internal control. The SDLC identifies the sequence of activities required in the systems development process and throughout the useful life of an application.

These elements are important to any well-defined systems development process. Each application system has a life cycle, based on a specific series of functions. The cycle is completed as replacement systems are implemented. Standards should be established and approved for each phase of the development process. Use of a corporate SDLC helps to better plan, implement, and maintain software systems. The following is a brief overview of the major components of an SDLC (Figure 12-1).

Figure 12.1
Waterfall System Development


Plan
Analyze
Design
Develop
Integrate & Test
Support
SHORT FALLS
• Analysis & design material convey limited understanding
• Resists most opportunities for change
• End Users not adequately involved
• First view of the software late in the life cycle where changes are more expensive

Project Initiation – Plan

The initial phase of the systems development process addresses conceptual changes and determines the feasibility of pursuing further development. A feasibility study should be performed to identify the expected costs and benefits of developing a specific system. Alternatives to internal application development, such as purchasing software or developing a nonautomated system, should be considered in preliminary evaluations.

Requirements Definition – Analysis

Once a project has been evaluated and approved for development, system analysis and design (involving a variety of complex tasks) proceeds. The requirements definition converts the system concepts into detailed specifications. The analysis and specifications must clearly identify user needs and expectations within the proposed application. This will facilitate development during the design and programming phases and should assist in evaluating vendor software products.

System Design, Development, Test, and Support

Systems design involves the conversion of user requirements into specifications for programming and implementation. The involvement of users is crucial for ensuring that the application design meets their requirements. Systems design techniques vary greatly. However, they should detail the sequence of program flow, the files to be used, the reports to be produced, and the controls to be built into the system. These standards also should identify the content for design documents and the procedures for review and approval. Design documents should include:

- A system narrative identifying the objectives and major functions to be performed.
- A system flowchart identifying the programs, data files, and reports.
- Record layouts showing the content for each data record and file in the system.
- Data element definitions describing each data field.
- Report layouts detailing the format and content for each report produced.
- Screen formats showing the fixed fields and those that are user-modified, including applied edit and control functions.
- Program specification for each program detailing program objectives, files required, and edit routines.
- A test plan validating that programs function properly and produce the expected results.
- A project schedule establishing expected benchmarks for various systems development phases.

The detailed system design can be considered the most important phase of an SDLC. It should represent the approved plan or contract between the user and systems development management.

SOFTWARE ACQUISITION

Many information systems applications for financial institutions have been developed by software firms. Financial institutions of all sizes find that purchasing software is a cost-effective alternative to developing software systems in-house. Volume sales allow software vendors to provide powerful and flexible products at competitive prices.

Vendors also can provide the necessary technical expertise for continued product support. However, software purchases should be analyzed carefully before any commitment is made. Written standards should exist for evaluating vendor software and for guiding the selection process. Since several vendors should be assessed, a request for proposal (RFP) is generally used to analyze a vendor's capability to meet the needs of the financial institution. An analysis of software products should consider:

- *System requirements* – The institution should identify user needs and expectations for the new system.
- *Systems specifications* – Input/output requirements, interfaces to other systems, and possible future enhancements should be defined.
- *Impact analysis/capacity planning* – The impact that the software will have on existing systems and hardware capacity should be evaluated.
- *A cost/benefit analysis of alternative purchase* – The capabilities of alternative software packages should be evaluated to determine how the proposed system specifications can best be met. A final decision should be reached only after alternative packages are considered.
- *Software support* – The institution must identify responsibility for software support. Many vendors offer software support and maintenance as part of their application system package. Vendors have varying system development processes and techniques which can be a factor in case of maintenance. When contracting for this service, the financial institution must assess both the technical and financial capabilities of the vendor.

If the system is to be supported in-house, the institution must assess its own technical resources and identify:

- *Financial condition of the vendor*s – A vendor should have the financial stability to continue in business and support its products as contracted. The institution should continue to monitor the financial condition throughout the contract term. If financial stability is in doubt, alternatives must be developed to reduce any adverse impact from a loss of vendor service.
- *A well documented system* – A vendor should provide sufficient documentation to allow the user to understand how the software works and the techniques and methods the software uses. This is especially important if program changes are to be performed by the purchasing institution. The format and content of this documentation probably will not be written according to the financial institution's information systems standards. The documentation of program packages under consideration must be reviewed prior to purchase to determine if the product generally meets the financial institution's minimum documentation standards.

  All changes made to application programs and operating systems must be documented according to prescribed standards. The financial institution must have staff with sufficient technical expertise to maintain the software and the documentation. Some vendors provide documentation on magnetic media and this information may be loaded into the on-line documentation system for reference purposes. A potential problem in documentation can arise if the purchased package requires modification before it is installed. It must be determined up front who will modify the documentation with the vendor and/or the purchasing institution involved in the process.
- *Escrow agreements* – Many vendors do not release the source code to the purchaser. This is intended to protect their system's integrity and copyright. The application system is installed in object code. An alternative to receiving the source programs is to establish an escrow agreement. In this agreement, which should either be part of the service contract or exist as a separate document, the financial institution would be allowed to access source programs under certain conditions, such as discontinued product support or financial insolvency by the vendor. Adequate programming and system documentation also should be required. A third party would retain these programs and documents in escrow. Financial institutions should determine periodically that the source code maintained in escrow is up-to-date. This can be done by a third party independently verifying the version number

of the software.

- *Training* – The financial institution's personnel training requirements should be clearly recognized. Training should cover such functions as system operations, problem resolution, input/output controls, file backup, and system support. Vendor responsibility for training should be stated in writing and clearly understood by both parties.

Regardless of whether purchased or developed in-house, the cost of software should be amortized over a reasonable period of time recommended by the institution's accountants.

PROGRAMMING STANDARDS

Although creativity in programming is both necessary and desirable, certain standards are needed to control the development of efficient software. A variety of programming techniques may be employed for this purpose. Standards should establish controls and procedures that must be followed during the programming phase. Although it is not possible to dictate the use of a specific technique in a certain program, all programs should contain some of the following controls:

- Accumulating detailed data and proving them to established control totals. Debit/credit totals, batch totals, and item counts are examples of this type of program control.

- Verifying data input in initial program editing. This allows corrections before erroneous data generates invalid results. This is possible for most applications.

- Using file labels for tapes and disk files to identify specific files, creation dates, and other information that establishes unique identity. This technique guards against invalid processing caused by computer operators selecting and mounting incorrect data files. To be effective, this control must be accompanied by an operational procedure that restricts the operator's authority to override programmed file label checks.

- Embedding edit routines in program code to verify specific data fields. Some examples are:

  - Sequence checks that verify that an item is in proper order for processing.

  - Overflow checks that prevent a data item from entering a storage field too small to accept it.

  - Format checks that verify that data in certain areas are in proper mode (numeric, alphabetic, or alphanumeric).

  - Reasonableness checks that compare data against known factors about the data items or classes of data.

  - Completeness checks that test to see if entries appear in fields that cannot be processed blank.

  - Range checks that compare a data item against set parameters.

  - Check digits that are generated as mathematical functions of the remainder of the field i.e., they permit program computation of the validity of data in input fields.

  - Data items that can be compared in different files or matched against prior period data items to determine reasonableness.

Many types of data integrity checks exist. They are useful in controlling data errors and should be included in applications programs. Program documentation should include a listing of all controls. Such controls include:

- *Restart capability* – Typically, specific stages of processing are identified. Checkpoints are established between processing stages and restart points. Data processed before a restart point is saved. Processing can be resumed at the checkpoint rather than starting from the beginning.

- *Transaction logging* – For on-line applications, the logging of all transactions processed or reflected by input programs provides a complete audit trail of actual and attempted entries. The log can be stored on tape or disk files for subsequent analysis.

- *Standardized program routines* – Many standard routines can be devised and merely embedded in applicable programs to avoid programming duplication.

- *Exception reporting* – Parameters are included in programs to identify unusual or nonstandard transactions. These uncommon transactions should be captured and recorded on exception reports for subsequent analysis. Exception reporting is a common method of control.

TESTING STANDARDS

As in the design phase, testing is an important phase of the programming process. Here the design and overall reliability of the application system is proven. Tests should be conducted using predetermined data under controlled conditions. This ensures that data will be processed correctly, and reliable output will be produced in the desired format. Strict standards should be developed to govern the testing process. Standard testing procedures should require:

- *A documented test plan* – This identifies specific files, data fields, calculations, and processing routines to be tested. It also should state the test periods e.g., daily and end-of-month.

- *A parallel test of new application systems* – This output can be compared with that existing to determine processing validity.

- *Independent verification of test results by user representatives* – Often users will assume complete responsibility for preparing test data and conducting tests. This is commonly referred to as user acceptance testing. Final test results should be reviewed by all affected parties, including computer operations personnel.

- *A restriction against the use of live files in testing to prevent destruction or alteration of live data* – A copy of a live file can be made to use for testing. Stress/volume tests should be included to ensure that the system can function at peak periods of transactions.

- *The simulation in test data of all possible error conditions to ensure that the program effectively handles all situations* – Although it is difficult to anticipate every error condition, test standards should specify that all abnormal processing conditions be tested. Inadequate testing is often responsible for processing errors that materialize months after the application system becomes operational. Test data generators (software) can aid in developing complete test data.

- *A thorough test of any changes to existing programs may include regression testing (testing of unchanged code)* – Often a simple program change can trigger a malfunction in an apparently unrelated area of a program or system. Ideally, a complete test of all program conditions should be performed each time a logic change is made. Program changes made in haste are often responsible for processing malfunctions and unwanted expenditures of programmer time in researching problems. Also, if copies of live files are used for testing, precautions should be taken to avoid any mix-up between test results and regular output reports.

SYSTEM IMPLEMENTATION

An institution should have written standards for implementing software changes and new systems to ensure that they are ready for production. Implementation standards should require:

- Documented acceptance of the application system by users and computer operations department. This verifies that the programs have been tested thoroughly and are ready for production.

- Complete user and operator run manuals describing each of the processing steps. Manuals must be employed in application testing to identify confusing or invalid operational instructions. The quality and reliability of such manuals is vital to successful information systems operations.

- System training for all users and computer operators. They should be specifically prepared to handle all known abnormal processing and data error conditions. Training should focus on following written procedures.

- Complete documentation before implementation. Analysts and programmers often tend to delay completion of documentation in favor of beginning the next project. Although documentation is generated during the design phase, it must be completed before this phase.

System Evaluation

A post-implementation review should be conducted shortly after the application has begun to operate. The review is conducted primarily to assess the operational performance of the application beyond an initial shakedown period. All parties actively involved with its operation should be interviewed and specific problems noted. Additionally, the relative success of the project should be gauged by comparing planned and actual costs, benefits, and development time. The results of the review should be documented in a post-implementation evaluation report presented to management. Post-implementation reviews should include:

- *Planned development time and cost compared to actual time and cost* – Management should be aware of projects that dramatically exceed planned cost and time targets. Future project selection and planning should be based on past experience.

- *Design objectives* – It should be determined that the application functions according to design specifications.

- *Potential savings (if applicable)* – Actual cost savings over a controlled period should be compared to the original estimated savings for a similar period. If the planned benefits do not materialize, reasons should be reviewed.

- *Identification of operational problems* – After the application has been in use, operational problems not identified during testing are likely to appear. The post-implementation review can identify problems and suggest future improvements. These secondary modifications are often important in perfecting the system.

- *Non-monetary benefits* – Planned and actual benefits should be compared to assess the project's success.

SOFTWARE MAINTENANCE

Some modification of information systems programs will be necessary during the application life cycle. Changes may be required by program problems, variance in internal operations, competitive demands, or other factors, such as regulatory changes. Whatever the justification, program changes must be strictly controlled and documented to prevent fraudulent or inadvertent modification.

Program Change Control

Requests for program changes should be documented on a standard change request form. The form is used to describe the request and document the review and approval process. It should contain:

- Date of the change request.

- A control sequence number.

- Program or system identification.

- Reason for the change.

- Description of the requested change.

- Person requesting the change.

- Benefits contemplated from the change.

- Projected cost.

- Signed approval authorizing the change, including, at a minimum, the user, appropriate level of information systems personnel, and audit (at least for significant changes).

- Name of programmer assigned to make the change.

- Anticipated completion date.

- User and information systems approval of the completed program change.

- Implementation procedures (steps for getting the program into the production library).

- Implementation date.

- Audit review of change (if deemed necessary).

- Documented sign-off.

Request forms can be pre-numbered or assigned a control sequence number as they are used. They might be filed numerically or by application. However maintained, the request forms should provide an accurate chronological record of the description of all changes to production programs. These records enable:

- The user to track all requested changes accurately.

- Information systems management to monitor the number and status of changes and allocate them to proper cost centers.

- The audit department to review significant changes for authorization and conformance to established standards.

Copies of the program change forms ordinarily should be distributed to the requestor, the system user, and the programming unit. Depending on the internal control process, the audit department may either require a copy or have access to copies. Notification of program changes to the audit department should not constitute or replace an audit of the program change process.

Both the user and programming management should review and approve change requests. This allows each unit to be aware of pending changes and clarify the specific request. Requested changes should be screened before acceptance to determine alternate methods of making the changes, the cost of the changes, and time requirements for programming. A risk analysis model also could be applied.

Strict procedures should be established to control the movement of modified programs into the production environment. These procedures should identify:

- The security of the programs before production.
- The approval process required to promote programs to production.
- The personnel responsible for placing the programs into production.

Once program modifications have been completed, all program codes (load module, object code, source code, patch code, etc.) should be secured. This will provide some assurance that any program cataloged to the production environment is an unaltered version of the tested and approved program.

The program approval process should include verification of test results, inspection of the code, and verification that the source code and the object code match. Verification of test results should be done according to testing standards.

A list of modified programs should be compared to the authorized change documents to determine that only approved changes were implemented.

Programming personnel should not be responsible for moving programs into production libraries. This responsibility should be assigned to an independent quality assurance/production control function in larger financial institutions or to supervisory operations personnel in smaller ones.

Systems programmers must modify operating systems periodically to resolve operating problems or implement enhancements. Control procedures for modification of operating systems should parallel those for changes to application programs. This includes prior author-ization, documentation, and review of the change. The absence of sound controls and documentation can cause problems when new releases of operating systems are installed.

All program changes should be fully documented. The approved program change request form usually can provide most of the needed documentation. Revisions to program documentation should occur when a change is implemented.

Temporary Program Changes

Occasionally, the need for a program change arises that must bypass the normal change procedures. Such a change might be required to restore production processing. These immediate changes are usually called patches, program temporary fixes (PTF), or temporary program changes (TPC). Sometimes the program object code is changed directly. In other cases, temporary changes to the source code can be made in a separate version of the program. The use of such techniques should be strictly controlled to prevent unauthorized changes and to ensure that approved changes are made correctly. The following controls should be adhered to:

- All temporary changes should be approved in advance by supervisory personnel.
- If the source code is changed, altered instructions should be reviewed by a knowledgeable supervisor.
- A form should be used to document temporary changes, although it often cannot be completed until after the change is made. The form should identify the change, indicate the reason for the change, detail the instructions altered, identify who made the change, the date it was made, and provide a signature of authorization.
- After a temporary change is made, the normal program change procedure should be completed as soon as possible. In the event a patch or PTF is made, the source code must be altered and recompiled to replace the patched object code.

INTEGRATED SOFTWARE SYSTEMS

Application software that combines multiple banking applications and processing functions is known as integrated software. Many financial institutions are converting to integrated systems to meet competitive demands, improve timeliness of information, foster operational efficiency, track the cross-selling of available services, and ease the introduction of new products.

Conventional software applications, such as demand deposits and installment loans, are designed to operate as separate systems. Each application has its own master files, transaction files, and unique program logic. Usually, there is little or no shared data between applications. In addition, programs that provide processing instructions for common functions, such as calculations, accounting, record structuring, or printing are unique to each application.

To improve the efficiency of applications systems, integrated systems were developed. In more basic system designs, integrated software is used to link separate applications. In these systems, data common to each application is stored and accessed through a customer information file (CIF, may also be called a central information file). The records might include customer(s) name, address, account number, and balance. Some of the data is maintained directly in this CIF system. Other data is transferred daily from the processing applications to the CIF. By consolidating this data from separate applications, all customer relationships with the institution can be accessed quickly. This improves operations, increases controls, and allows better customer service. This design has been particularly beneficial to tellers, customer service representatives, and managers. This CIF concept differs from a data base management system (DBMS), since the files are still sequentially accessed. Another differentiation between CIF and DBMS is that CIF tells you the name and address of accounts, while DBMS stores the actual data.

Large Scale Integrated Systems

Certain fully integrated systems combine multiple applications into a single processing system. These large scale integrated systems (LSIS) require a complex design. The system architecture may be horizontal or vertical. Under a horizontal structure, applications such as deposits, loans, and general ledger are linked. Under a vertical structure, functions are linked. For example, a teller transaction is processed at the teller station, additional transactions and entries are generated for operating departments affected by that teller activity.

As part of the design, integrated systems often employ data base management technology. Their processing structure may use combinations of batch, on-line, or memo posting methods. See Chapter 25 for additional information on FFIEC SP-4: Supervisory Policy on Large Scale Integrated Financial Software systems.

Integrated software systems are advantageous, because they:

- Increase efficiency and reduce unit processing costs through standardized operating techniques.
- Eliminate redundancy in data files.
- Provide tools to increase product lines and customer relationships.
- Enable financial institutions to meet competition generated from forces outside the banking industry.
- Improve the timely delivery of information throughout the institution.
- Allow more informed management decisions.

Managing the Project

A commitment to an LSIS sets the course of an institution's technology, management information systems (MIS), and delivery systems for several years. Although large scale integrated systems offer substantial advantages, their inherent complexity requires more formal development and maintenance procedures, specialized audit techniques, and senior management commitment. These are essential to the successful implementation and operation of large integrated systems.

As financial institutions convert to integrated software systems, many experience considerable difficulties. Some institutions spend millions of dollars and years of conversion and implementation time, only to abandon the project.

To avoid these difficulties, factors which should be considered by management, a full commitment to this type of project include:

- The financial condition and viability of the vendors when evaluating systems.
- The financial impact on the institution, in terms of not only the initial cost, but also the time and resources required for successful installation.
- If the system uses real-time data base technology, the cost of data backup and recovery measures, as the entire data base must be backed up simultaneously.
- The time and capability of a backup computer to test the disaster recovery plan of an LSIS.
- Whether SDLC methodologies need to be modified because of the increased complexity of the software and the difficulty of assuring system integrity (seemingly simple program changes can have unpredictable results in a multi-application system).
- Whether audit and control issues are adequate for real-time integrated systems.
- Whether system access is difficult to control and monitor, especially when using dial-up access.
- Whether audit trails address system-generated transactions sufficiently.

Management Responsibilities

The decision to acquire or develop in-house LSIS should be preceded by a strong and independent management planning process. This should include a thorough assessment of existing software performance. Also essential is a detailed analysis of additional capabilities necessary to accomplish the institution's strategic business plan.

The complexity of the integrated software and its impact on the entire organization requires a commitment from senior management for the project to be successful. Responsibility for the conversion should be identified clearly and established at the senior management level.

The institution's board of directors should review the project often to maintain control over the complex process of implementation and to ensure completion within established parameters. Board reporting formats should be established at the beginning of the project. The board must continue its oversight responsibilities after implementation.

OPERATING SYSTEMS

Current generation computers differentiate between application programs and operating systems/supervisor programs. The operating system controls all input, output, and interrupts. It also controls program scheduling and hardware resource allocation.

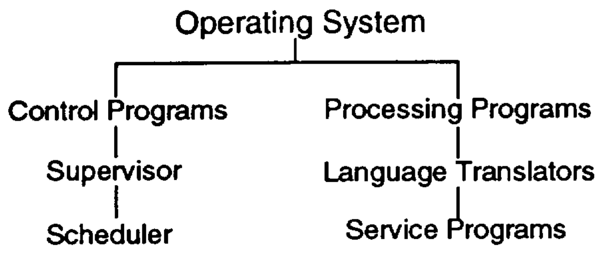
The operating system:

- Increases system throughput by overlapping input and output operations with processing.
- Improves user turnaround by permitting simultaneous processing of applications.
- Simplifies implementation of new applications by providing general purpose utility programs, thereby reducing the number of programs that must be written.
- Assists programmers by relieving them of the burden of writing logical and physical input/output control instructions.
- Improves overall operational efficiency and control by providing:
  - Job scheduling.
  - Analyses for job accounting and computer system use.
  - Control over programs stored in the computer.
  - Control over access to and use of the data files.
  - Control over use of all hardware resources.

Most operating systems contain a significant variety of optional features. Specific features can be selected for implementation in individual installations. Since some features change the processing and can either add to or overcome controls, a record of them should be kept.

An operating system generally consists of the following programs (Figure 12-2):
-
Control programs. The control programs provide control over a continuous series of jobs, the workflow within the system, and the input/output operations. Control programs consist of two basic components:

Figure 12.2
Sample Basic Elements of an Operating System



- The supervisor program controls how computer system resources are used. It coordinates all processing to assure that each program receives the services requested. The supervisor program normally responds through an interrupt system and a specialized set of privileged instructions.

- The scheduler program accepts job instructions, allocates input/output devices, initiates the execution of processing programs, and maintains communications between the system and the operator.

For efficiency, only the control program and a few of the most commonly used service programs will reside in memory at all times. The resident memory portion of the operating system is sometimes referred to as the nucleus. All other service programs reside in a library, usually on magnetic disk.

Processing programs. Each operating system provides a comprehensive set of generalized programs designed to assist the user. Some of the current programs are:

- Language translators to assist programmers by translating application programs written in a high-level language into machine instructions. The majority of mainframe computer manufacturers provide the following language translators:

  - COBOL. This is an acronym for COmmon Business Oriented Language. This language is best suited for commercial or accounting functions.

  - FORTRAN. This is an acronym for FORmula TRANslator. FORTRAN is an algebraic language and is well-suited for scientific and engineering applications.

  - BASIC. This is a simplified language similar to FORTRAN. It is an acronym for Beginners All-purpose Symbolic Instruction Code. It differs from FORTRAN, since it can function in an interactive or conversational mode.

  - RPG. This is an acronym for Report Program Generator. RPG was originally indented to permit the user to write simple report programs with minimum effort. It is now used as a stand-alone programming code (RPG-II and RPG-III).

  - Assembler. This is a low-level language. Although difficult to code, it may result in more efficient programs. The use of Assembler language allows bit level data access and usually makes all machine instructions available to the programmer.

  - C. Although primarily a microcomputer or workstation language, it is gaining popularity in DEC/VAX systems because of its portability. An extension of C called C++, an object oriented program, is expected to gain popularity in the future.

- Service programs to provide routines for commonly performed functions. Examples include linkage editors, sort/merge, library maintenance, data management, and manipulation. These programs can be divided into two basic groups:

  - System service programs. These programs are called by the supervisor in response to interrupts. The user may, in some cases, change priorities or provide parameters, but does not execute the program or routine. Some examples of system service programs are:

    Open/close macro instructions for data files.

Program check and memory dumps.

Data file access routines.

Routines for loading and executing programs.

Job accounting routines.

System error checking routines.

Routines for allocating memory.

Spooling routines for input and output.

Queuing routines for entry of jobs and assignment of priorities.

Communication programs in on-line environments.

– Utility programs. The following general purpose programs are initiated directly by the user:

  · Sort/merge programs for sorting and merging data files together.

  · Linkage editor programs to link programs that have been compiled or assembled by a language translator. (When a program is too large for memory, the linkage editor can be used to divide the program into overlays.)

- Data file manipulation, copy, or print programs.
- Program library maintenance routines.

PROGRAMMING PERSONNEL

Application Programmers

Application programmer activities should be clearly defined. Access to programs outside the programmers' individual responsibility should be restricted. Maintenance expertise should exist for each major application, and responsibility assigned to specific programmers. Programmers not assigned to maintain or develop specific programs should have access restricted only to such programs.

If application programmers are allowed to write programs in low-level languages, such as Assembler, they should not be permitted to execute instructions generally restricted to the operating system.

Programmer use of storage dumps should be restricted, so that only those persons authorized to use specific dumps have access to them. Storage dump listings and program source listings used by programmers should be secured when not in use and destroyed when no longer needed.

Systems Programmers

In many organizations, a distinct group of highly skilled programmers must maintain operating systems, teleprocessing control systems, data base systems, and develop supportive applications. Such programmers are generally known as systems programmers.

Larger data centers may refer to their systems programmers as the technical support group. This group serves as liaison with manufacturer system software support personnel, application programmers, and the information systems operations staff. Group responsibilities for computer operations support include:

- Controlling program libraries and data sets.
- Maintaining operating system software and data base software.
- Identifying access methods.
- Installing other support software for use by operations, systems, and programming personnel.
- Allowing add-on systems, such as ATMs, to function.

The complexity of the technical support activities hinders the effective monitoring systems programming efforts. Written standards should be developed and enforced to monitor and control all programming activities. A clear distinction between the application programming and systems programming duties should exist.

PROGRAM SECURITY

Strict security should be maintained over access to and use of computer programs with such security being physical and/or logical. Data security should be addressed prior to the installation of such a system. Existing data security systems may not be adequate for a complex integrated system, particularly one using on-line real-time processing. Each individual function should be controlled, e.g. access controls, file main-tenance, inquiry, and new accounts. Reference SP-4. Procedures should restrict unauthorized access to:

- Application programs.
- Operating systems programs.
- Data files.
- Documentation.
- Computer equipment.

In addition, periodic supervisory review of activity logs, time records, and other reporting techniques can be used as a monitoring technique. Specially designed software programs may be used to flag exceptions or changes to libraries.

At least three proprietary program librarian systems offer additional techniques to monitor program security. These systems can protect source and object libraries, maintain program version numbers, produce system activity logs, retain a chronological record of actual program coding changes, and restrict operator access to unauthorized functions. Since program development or modifications also can be performed from microcomputers after downloading program files from the mainframe, at least one librarian system offers increased monitoring of accountability for this type of end user access.

Systems also can alter or delete programs or data. This capability is contained in utility programs that are usually supplied by the manufacturer. Systems utility programs are valuable tools when used during program debugging, file maintenance, cataloging, or even daily operations of the data center. However, certain programs can be used to alter core storage, data files, and object code; enter the supervisor state; and catalog, purge, and rename programs. Such action can be justified occasionally when data becomes garbled or can be unjustified when an improper action is contemplated. Most computer manufacturers supply these programs as part of the operating system. Programs, such as DITTO, IMASZAP (Superzap), IEBGENER, MSHP, IEBRENAM, IEBUPDAT and IMASZOT, should be controlled to prevent unauthorized use. Manufacturer utility program manuals can be consulted to determine programs to be controlled.

Unauthorized use of system utility programs can be controlled in several ways. These controls, however, will not be effective, if they unnecessarily impede operations. System utilities can be controlled by:

- Removing utility programs that have data or program altering capabilities from the system catalog. Place these specific utilities, which are on machine-readable media, under the physical control of the shift or operations supervisor.

- Installing a password system on all program libraries, including the system utility library. If password protection is used, measures must be taken to control access to passwords. Passwords should be changed periodically.

- Using automated librarian systems. Several automated librarian systems that provide program security are available from equipment manufacturers and software vendors. Such programs restrict access to the program library. They can produce daily reports identifying each program that was accessed and any program changes that were made.

Even the most sophisticated program security system will fail, if management does not establish, implement, and maintain adequate internal and operating controls. Standards and procedures for all aspects of program development, processing, and maintenance are necessary. These must identify control responsibilities and processes for documentation, review, and approval of various programming activities. Supervision of these functions is imperative for establishing and maintaining program integrity.

DOCUMENTATION STANDARDS

The value of documentation is derived from its accuracy, timeliness, and completeness. Well-documented systems are easier to maintain and convert to new systems. Good documentation also facilitates the rotation of personnel/separation of duties, and supports the continuity of operations in the event of key personnel turnover. Program documentation is often time consuming and regarded as non-creative. However, it is the most convenient method of determining program functions. Furthermore, inadequate documentation can often lead to a lack of administrative control.

There are no universal documentation standards. The American National Standards Institute (ANSI) has approved standard number 1063 developed by the Institute of Electrical and Electronics Engineers, Inc. (IEEE) which applies to software user documentation; however, this standard does not apply to design, development, or changes which involve modification of source code. Effective documentation procedures require the involvement of financial institution and information systems management.

As a starting point, common practices in the financial institution information systems industry should be reviewed. The development of documentation standards could be a process of adaptation rather than origination.

The development of documentation standards will require analysis of the financial institution's current information systems environment and projections of future requirements. After the general scope of the documentation is defined, the format must be organized in a logical manner. Various methods and procedures creating documentation should be investigated. Each step of the process should be reviewed and evaluated. When documentation is complete, plans should be implemented to update and revise it as the financial institution's needs become more sophisticated and diverse. The standards and procedures manual should contain standards for documentation. If the institution allows end user modifications of programs from microcomputer or LAN systems, the institution also should specify the level of user documentation which is required.

Regardless of the method used by the installation to organize its operating standards, procedures and requirements for developing and retaining program documentation must be included. The essential elements of program documentation follow.

APPLICATION SOFTWARE DOCUMENTATION

Each application in production should be supported by a complete set of program documentation. For each application, documentation should contain a problem definition, application system and program descriptions, operator and user instructions, and records of acceptance of the system. Access to the documentation should be restricted. Personnel should have access to only those sections of the documentation directly related to their job functions.

The following further describes documentation sections:

- Problem definition – Provides a clear, logical and formal record of the problem to be solved. The contents of the problem definition may include many of the concepts in the below example. (Figure 12.3)

  - Project background and project request (usually by the potential user).
  - Minutes of steering committee meetings or copies of policy decisions relating to the proposed job.
  - A problem statement defining the purpose of the proposed program.
  - Estimated and incurred costs.
  - Programmer-hours forecast and incurred.
  - Basis for decisions.

- Application system description – Provides an overview of the total application and ties together the individual computer programs within the application system. The application system description may be subdivided as follows:

  - System flowchart. Indicates the source and nature of the input, appropriate automated and manual operations, and the nature and disposition of output. The system flowchart illustrates the flow of the work through the application system (Figures 12.4 and 12.5).
  - System narrative. Describes the general outline of the application system, the related environment or computer system in which the program will operate, the operations it affects, its users and the interrelationship of its uses.

  Record layout schematics. Reflect the format of data items on cards, magnetic tapes, paper tapes, or disk tracks. Disk and tape layouts contain additional information on record size, blocking factors and the layout and contents of internal labels (Figures 12.6).

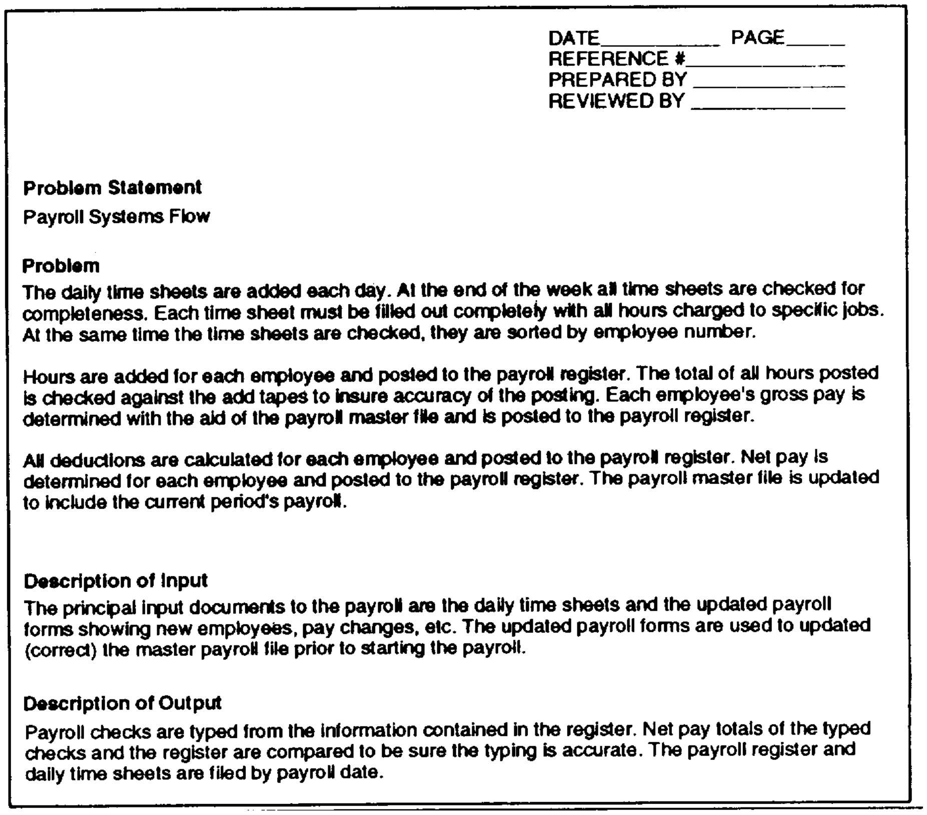
- Print layouts. Reflect data items output to print, showing columns, column headings, number of lines, spacing, and report titles.

- File description. Reflects the collection of related records that form a file for the information system processing application. A file description might logically describe an interim file, such as the sorted demand deposit transaction file, in order to further define requirements for processing. It also may logically describe the demand deposit master file for the same reason.

- Special code. Identify various types of transactions, activity accounts or other special indicators.

- Control function. Describe the accounting and operating controls. On-line systems should include a complete security narrative and a list of users having authority to inquire and update. Methods needed to maintain security also should be described. Restart and recovery procedures should be detailed if applicable.

Figure 12.3
Sample Problem Definition Statement



DATE__________ PAGE_____
REFERENCE #______________
PREPARED BY ______________
REVIEWED BY ______________

Problem Statement

Payroll Systems Flow

Problem

The daily time sheets are added each day. At the end of the week all time sheets are checked for completeness. Each time sheet must be filled out completely with all hours charged to specific jobs. At the same time the time sheets are checked, they are sorted by employee number.

Hours are added for each employee and posted to the payroll register. The total of all hours posted is checked against the add tapes to insure accuracy of the posting. Each employee's gross pay is determined with the aid of the payroll master file and is posted to the payroll register.

All deductions are calculated for each employee and posted to the payroll register. Net pay is determined for each employee and posted to the payroll register. The payroll master file is updated to include the current period's payroll.

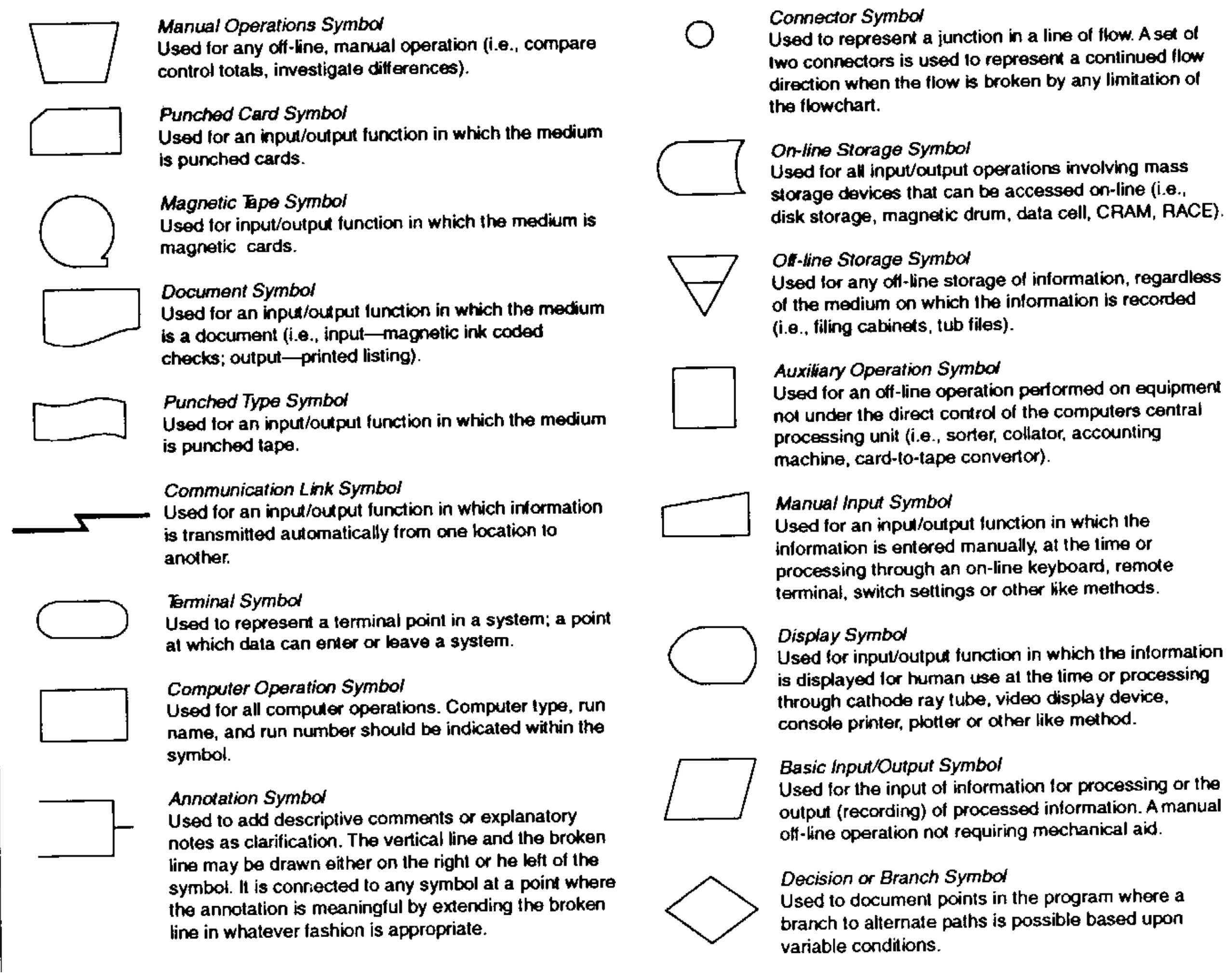
Description of Input

The principal input documents to the payroll are the daily time sheets and the updated payroll forms showing new employees, pay changes, etc. The updated payroll forms are used to updated (correct) the master payroll file prior to starting the payroll.

Description of Output

Payroll checks are typed from the information contained in the register. Net pay totals of the typed checks and the register are compared to be sure the typing is accurate. The payroll register and daily time sheets are filed by payroll date.

Figure 12.4
Sample Flowchart Symbols



Manual Operations Symbol
Used for any off-line, manual operation (i.e., compare control totals, investigate differences).

Punched Card Symbol
Used for an input/output function in which the medium is punched cards.

Magnetic Tape Symbol
Used for input/output function in which the medium is magnetic cards.

Document Symbol
Used for an input/output function in which the medium is a document (i.e., input—magnetic ink coded checks; output—printed listing).

Punched Type Symbol
Used for an input/output function in which the medium is punched tape.

Communication Link Symbol
Used for an input/output function in which information is transmitted automatically from one location to another.

Terminal Symbol
Used to represent a terminal point in a system; a point at which data can enter or leave a system.

Computer Operation Symbol
Used for all computer operations. Computer type, run name, and run number should be indicated within the symbol.

Annotation Symbol
Used to add descriptive comments or explanatory notes as clarification. The vertical line and the broken line may be drawn either on the right or he left of the symbol. It is connected to any symbol at a point where the annotation is meaningful by extending the broken line in whatever fashion is appropriate.

Connector Symbol
Used to represent a junction in a line of flow. A set of two connectors is used to represent a continued flow direction when the flow is broken by any limitation of the flowchart.

On-line Storage Symbol
Used for all input/output operations involving mass storage devices that can be accessed on-line (i.e., disk storage, magnetic drum, data cell, CRAM, RACE).

Off-line Storage Symbol
Used for any off-line storage of information, regardless of the medium on which the information is recorded (i.e., filing cabinets, tub files).

Auxiliary Operation Symbol
Used for an off-line operation performed on equipment not under the direct control of the computers central processing unit (i.e., sorter, collator, accounting machine, card-to-tape convertor).

Manual Input Symbol
Used for an input/output function in which the information is entered manually, at the time or processing through an on-line keyboard, remote terminal, switch settings or other like methods.

Display Symbol
Used for input/output function in which the information is displayed for human use at the time or processing through cathode ray tube, video display device, console printer, plotter or other like method.

Basic Input/Output Symbol
Used for the input of information for processing or the output (recording) of processed information. A manual off-line operation not requiring mechanical aid.

Decision or Branch Symbol
Used to document points in the program where a branch to alternate paths is possible based upon variable conditions.

- Program description – Describes the details of the individual programs and should include:
  - Program flowcharts (if applicable). Present the logic and flow of the program, subroutine or module within a major program. This documentation also may be termed a block, a logic diagram or logic chart. Flowcharts may be necessary for complex sections of programs and for programs written in low-level languages. Automated flow chart packages are available which will either automatically flow chart a program based on source code or will enable the programmer to flow chart a program without the need for drawing the chart manually.
  - Decision tables (if applicable). Supplement or replace flowcharts for description and documentation of more complex programs. The decision table is a tabular representation of the program logic that specifies all conditions and action relationships according to a set of user defined rules (Figure 12.7).
  - Table description. Present sets of data items

stored in tables for reference at various points in a program. Description of such tables and their value are a necessary part of documentation.

- Flags and switches. Describe program tests for specified conditions which are either present or changed, depending on the flow of instructions dictated by the logic of the program.

- Program change and system modification forms. Describes a change to an application program, application system or the operating system showing the result, benefits to the user and proper authorization. A detailed explanation should be maintained (Figures 12.8 and 12.9).

- Program listing. Should contain nontechnical comments and meaningful data names. Program listings and narratives are the most basic items in program documentation. Many of the other elements of program documentation can be derived from the program listing. The listing should be updated when programs are changed. With paper reduction generally mandated, institutions are either using microfiche for listings or retaining source listings on-line with appropriate access controls in effect.

- Programmed controls. Describes all editing performed by the program.

• Operator instructions – Explain how a particular job is performed and how operators should respond to certain system requests or when halt conditions

Figure 12.5
Sample System Flowchart

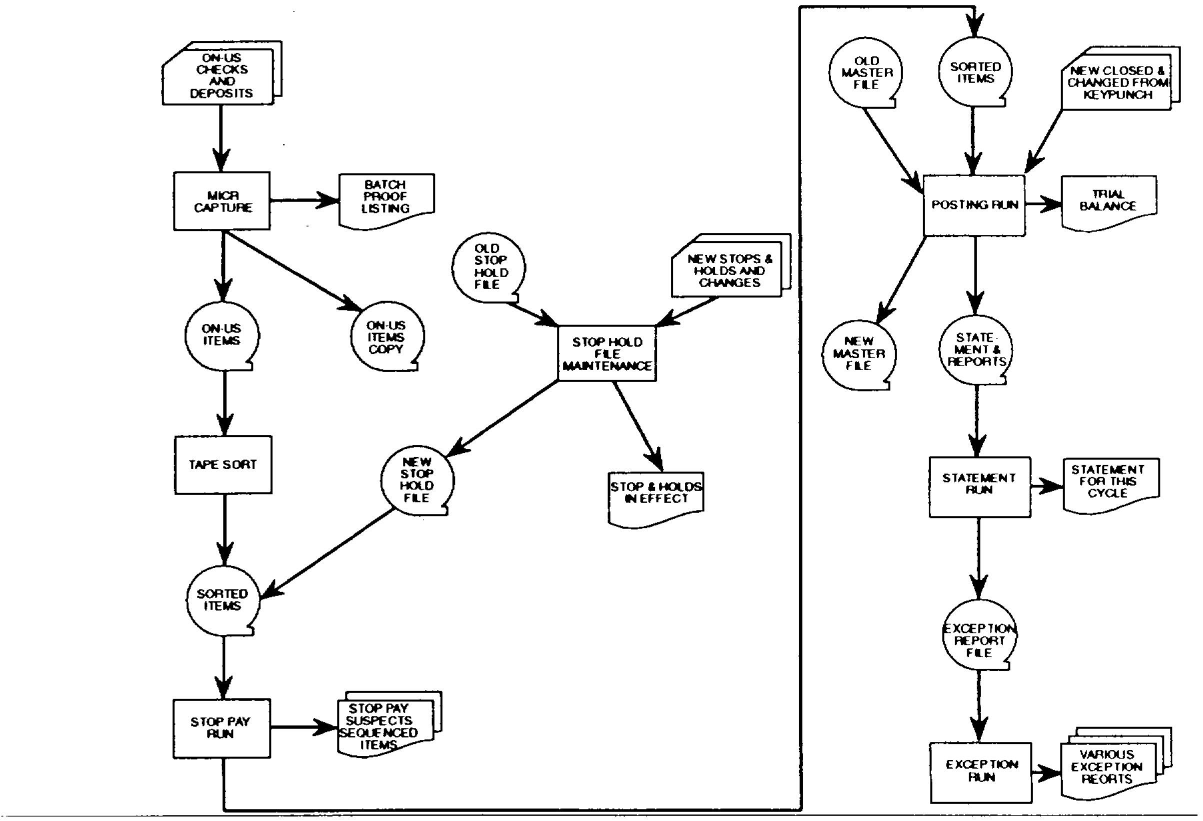
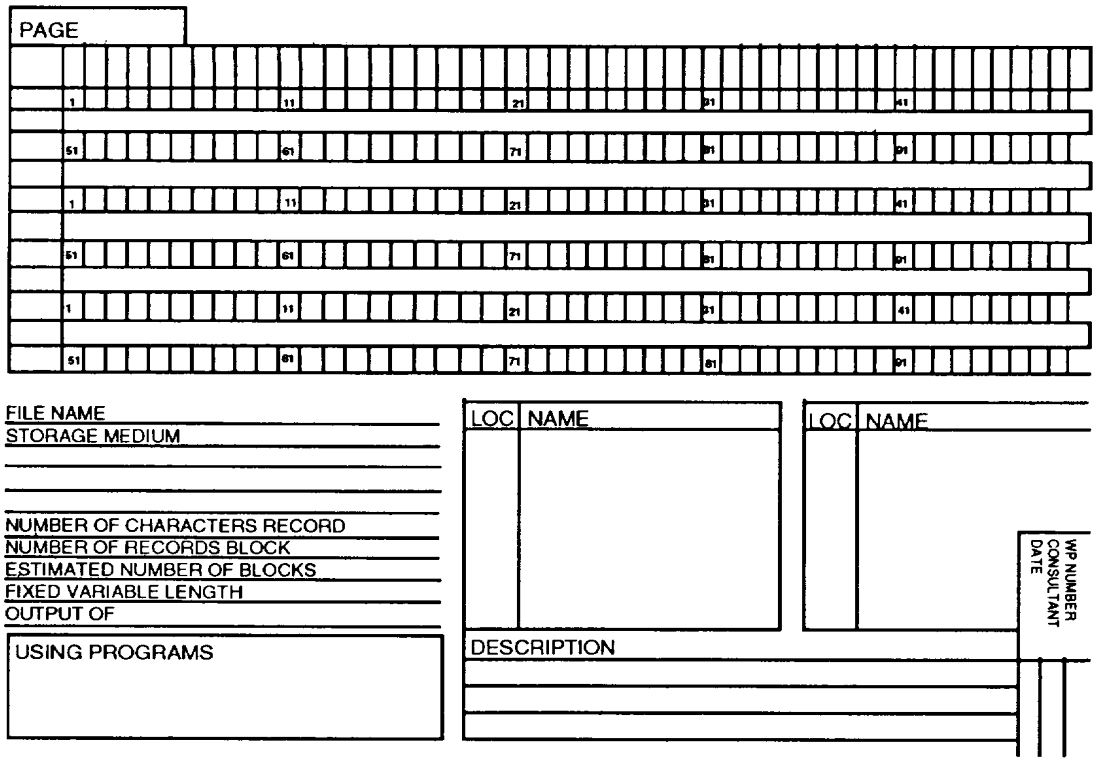


Figure 12.6
Sample Record Layout


PAGE

1 11 21 31 41
51 61 71 81 91
1 11 21 31 41
51 61 71 81 91
1 11 21 31 41
51 61 71 81 91

FILE NAME
STORAGE MEDIUM
NUMBER OF CHARACTERS RECORD
NUMBER OF RECORDS BLOCK
ESTIMATED NUMBER OF BLOCKS
FIXED VARIABLE LENGTH
OUTPUT OF
USING PROGRAMS

LOC	NAME	LOC	NAME

DESCRIPTION

WP NUMBER
CONSULTANT
DATE

occur. This documentation should include only information pertinent to the computer operator's function; program documentation such as source listings, record layouts, and program flowcharts should not be accessible to the operator. Operator instructions should be thorough enough to permit an experienced operator who is unfamiliar with the application to run the program successfully without assistance.

- User manuals (see the section on user manual documentation).

- Record of acceptance – Documents the formal acceptance of the completed application program modification. At a minimum, formal acceptance should be obtained from:

  - User department. Should indicate that test results have been reviewed and that the system meets specifications.

  - Information systems department management. Should indicate review of test results and control functions.

  - Audit department. Also should indicate review of test results and audit/control functions.

  - Information systems operation. Should indicate review of run instructions and acceptance of any additional hardware demands and run time.

  - Documentation librarian function. Should indicate review of documentation for completeness and compliance with standards.

When developed or modified in-house, operating system software must be supported by comprehensive programmer and user documentation. Regardless of the source of operating system and other support software, standards and procedures for maintaining such documentation must be developed by the financial institution.

MAINTENANCE OF DOCUMENTATION

The documentation librarian function is responsible for the control, retention, storage, and distribution of master documentation files. The exact procedures

Figure 12.7
Sample Decision Table

DATE ______________ PAGE ______________
REFERENCE NO. ______________________
PREPARED BY ________________________
RECEIVED BY ________________________

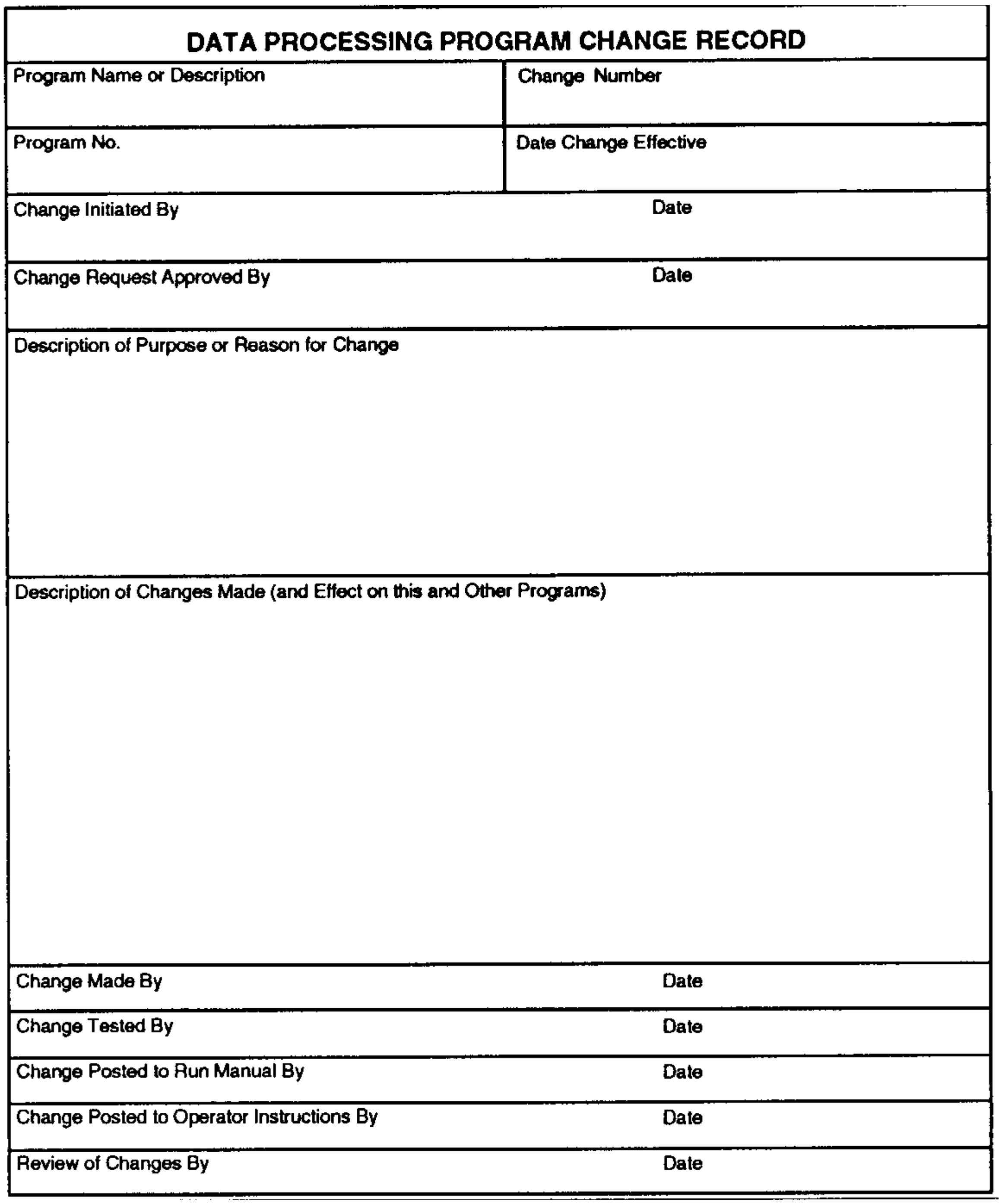
DECISION TABLE ___________________________

____

STUB	ENTRY																			
CONDITION	RULE NUMBER																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ACTION	RULE NUMBER																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

PAGE ____ OF ____

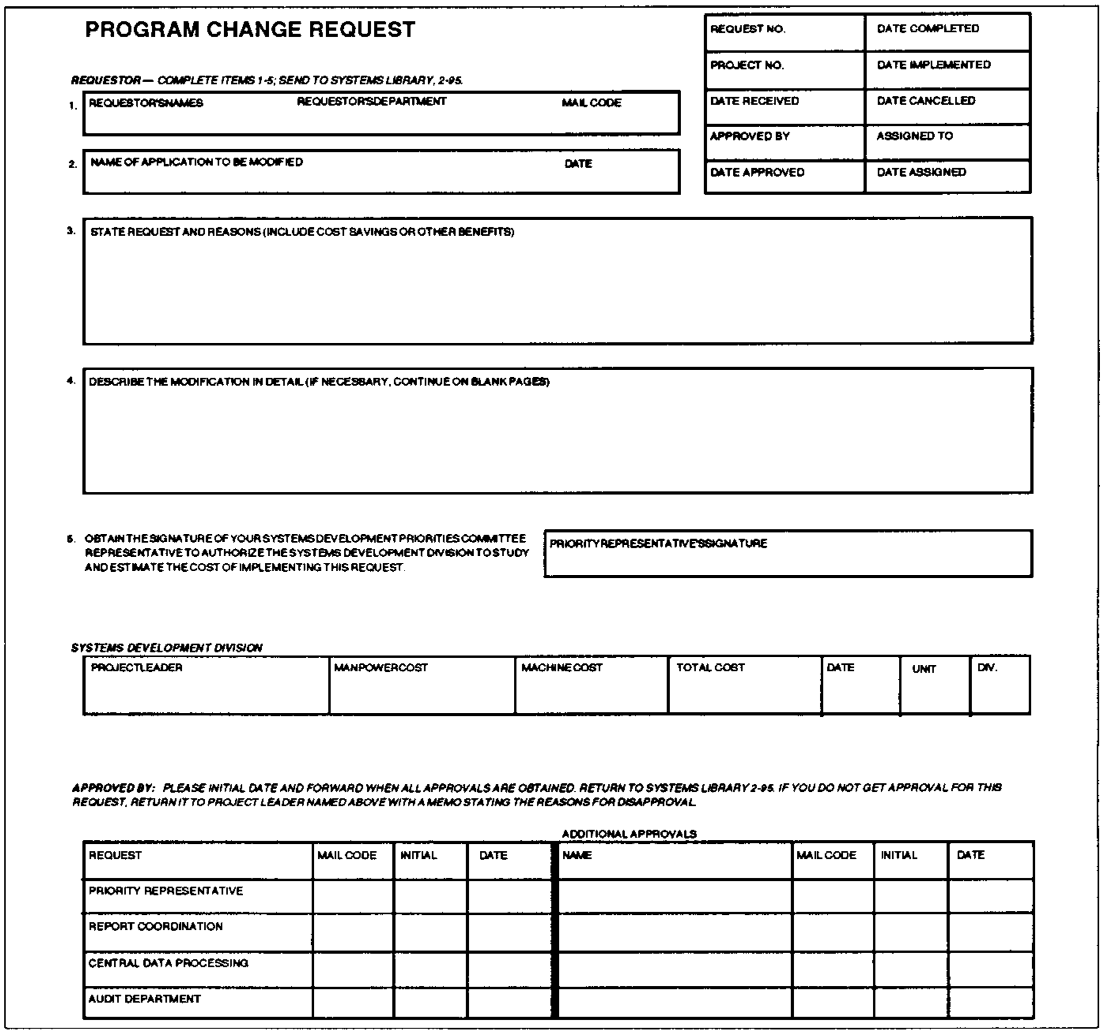
Figure 12.8
Sample Program Change Request



DATA PROCESSING PROGRAM CHANGE RECORD

Program Name or Description	Change Number
Program No.	Date Change Effective
Change Initiated By	Date
Change Request Approved By	Date
Description of Purpose or Reason for Change	
Description of Changes Made (and Effect on this and Other Programs)	
Change Made By	Date
Change Tested By	Date
Change Posted to Run Manual By	Date
Change Posted to Operator Instructions By	Date
Review of Changes By	Date

Figure 12.9
Sample Change Request Form



PROGRAM CHANGE REQUEST

REQUEST NO.	DATE COMPLETED
PROJECT NO.	DATE IMPLEMENTED
DATE RECEIVED	DATE CANCELLED
APPROVED BY	ASSIGNED TO
DATE APPROVED	DATE ASSIGNED

REQUESTOR — COMPLETE ITEMS 1-5; SEND TO SYSTEMS LIBRARY, 2-95.

1. REQUESTOR'S NAMES REQUESTOR'S DEPARTMENT MAIL CODE
2. NAME OF APPLICATION TO BE MODIFIED DATE
3. STATE REQUEST AND REASONS (INCLUDE COST SAVINGS OR OTHER BENEFITS)
4. DESCRIBE THE MODIFICATION IN DETAIL (IF NECESSARY, CONTINUE ON BLANK PAGES)
5. OBTAIN THE SIGNATURE OF YOUR SYSTEMS DEVELOPMENT PRIORITIES COMMITTEE REPRESENTATIVE TO AUTHORIZE THE SYSTEMS DEVELOPMENT DIVISION TO STUDY AND ESTIMATE THE COST OF IMPLEMENTING THIS REQUEST.

PRIORITY REPRESENTATIVE'S SIGNATURE

SYSTEMS DEVELOPMENT DIVISION

PROJECT LEADER	MANPOWER COST	MACHINE COST	TOTAL COST	DATE	UNIT	DIV.

APPROVED BY: PLEASE INITIAL DATE AND FORWARD WHEN ALL APPROVALS ARE OBTAINED. RETURN TO SYSTEMS LIBRARY 2-95. IF YOU DO NOT GET APPROVAL FOR THIS REQUEST, RETURN IT TO PROJECT LEADER NAMED ABOVE WITH A MEMO STATING THE REASONS FOR DISAPPROVAL.

				ADDITIONAL APPROVALS			
REQUEST	MAIL CODE	INITIAL	DATE	NAME	MAIL CODE	INITIAL	DATE
PRIORITY REPRESENTATIVE							
REPORT COORDINATION							
CENTRAL DATA PROCESSING							
AUDIT DEPARTMENT							

performed depend on the size and complexity of the data center and if the documentation is maintained on paper, on-line, or a combination.

The extent of the librarian function depends on the scope of information systems activities, the frequency of reference to library materials, whether processing functions are centralized or decentralized, the internal structure of the information systems department, and the medium on which the documentation is maintained. Specific librarian functions may include:

- Review of documentation at system development time to ensure adherence to standards and appropriate authorization of exceptions.
- Control and storage of acceptable documentation.
- Handling of documentation revisions.
- Notification and distribution of documentation to interested or authorized parties.
- Current maintenance of all documentation.

In small installations, the documentation library area may consist of a few filing cabinets in the programming area, maintained by a clerical person or the programming supervisor. In larger installations, it may be a separate area staffed by a full-time librarian particularly if paper documents are required for all forms of documentation. Such libraries may employ control and access procedures similar to those exercised over data file libraries. Now that many institutions are using on-line facilities for the bulk of documentation, filing cabinets may be appropriate to maintain documentation. However, in all instances, the documentation library or storage areas should be secured to prevent unauthorized access.

Documentation must be kept current. Standards for documentation must specify the authority and techniques for maintenance and control; responsibility for maintaining program documentation must be assigned.

Actual techniques for revising the documentation will depend on its structure and changes to it. Documentation can be revised either by programmers or by technical writers. Explicit rules for amendments must be established to prevent needless paper accumulation. Accordingly, provisions must be made for:

- Changes which can be made directly on existing paper documentation (if applicable) without making the end result illegible or unintelligible.

- Additions which amplify, clarify, or augment existing documentation without making the present content obsolete.

- Changes which are a complete or a partial replacement of existing documents.

- Changes which can be made to on-line documentation.

Documentation plays such a vital role that procedures to duplicate essential portions should be established. The duplicates should be retained in an off-premises storage facility and must be periodically updated. If documentation is on-line, the back-up of this information on magnetic media can be accomplished easily.

USER MANUAL DOCUMENTATION

User manuals enable the user to understand, approve and operate the system. Lack of user involvement may result in the user:

- Instituting inadequate remedies to problems, thus negating certain controls within the system.

- Contending that the whole system is defective.

- Placing complete reliance on the information systems department for controlling the operation of the application.

Continuing user involvement is of primary importance; therefore, user aids or manuals are essential aspects of program documentation. The responsibility for writing user manuals and user manual standards should be established formally.

A user manual could be divided into two main areas: input and output. The manual should be written in simple, nontechnical language, and not directed to a particular audience. When necessary, illustrations should show proper completion of forms, workflow, and descriptions and explanations of reports generated. When forms are illustrated, they should contain representative sample entries and instructions keyed to those entries. Every transaction code should be described in sufficient detail to allow the user to fully understand how to use the application system. Output reports should include detailed explanations of special codes, flags and reconcilement procedures necessary to balance the report user controls. The output reports need to be reviewed periodically to determine if they are being used as intended, or if they should be modified or discontinued.

PURCHASED SOFTWARE DOCUMENTATION

A prime consideration in any purchased package is documentation. If a package is not well documented, the user will not understand the techniques and methods used. The package may be less effective and efficient as well as difficult to maintain if program changes are to be performed by the purchasing institution. The format and content of this documentation, however, probably will not be written according to the financial institution's information systems standards. The documentation of program packages under consideration must be reviewed prior to purchase to determine if the product generally meets the financial institution's minimum documentation standards. Such precautionary measures also should be applied to the purchase of standard utility programs, such as sort or merge programs.

Purchased software will be subject to subsequent modification by the user or vendor. All changes made to application programs and operating systems must be documented according to prescribed standards. When contracts are written to include continuing maintenance of the software by the vendor, the vendor must have the technical and financial ability to perform this task. This is especially important for installations that do not have an adequate programming staff. Some vendors provide documentation on magnetic media and this information may be loaded into the on-line documentation system for reference purposes. A potential problem in documentation can arise if the purchased package requires modification before it is installed. It must be determined up front who will modify the documentation with the vendor and/or the purchasing institution involved in the process.

DATA BASE MANAGEMENT SYSTEMS

Data base management systems are playing an increasing role in modern information processing. A data base is a collection of nonredundant, interrelated data elements that are acted upon by one or more application programs. Data elements are organized to form distinct data files that are either relational, hierarchial, or network interconnected in nature. Data files are identified by a unique key, such as an account or employee number, and may contain pointers or indices to all of a customer's accounts. Data is accessed through a common software controller, commonly referred to as a data base management system (DBMS). Generally, the DBMS is provided by the manufacturer or an independent software vendor. Through the controller, an application program can access any relevant record regardless of its location or format. The data manipulation language (DML) interfaces between the application program and the DBMS to perform various functions such as opening a file, inserting a record, or changing data. An example of DML is IBM's Structured Query Language (SQL) which is used with IBM's DB2 relational database.

The sophisticated features of a data base, if properly designed, can greatly benefit a financial institution. However, a data base also can contain some disadvantages management should be aware of. Advantages and drawbacks include:

Advantages

- Reduction of data redundancy. In a multiple file system, the same data items are often stored in separate files. Such storage creates an updating problem since each file must be separately updated whenever a change to the data occurs. To the extent possible, a data base system only will store each item once, thus eliminating multiple updates and saving storage space.

- Program and data independence. Previously, it was necessary to define format and storage characteristics of data within each program. A data base system separates the physical storage and accessing functions. Therefore, the data base can change and grow without necessitating an update to each application program. In addition, the programs deal with data content rather than location. This independence can facilitate moving the applications from one computer to another.

- Standardized data definition and documentation. These functions are standardized by using a data dictionary and controlled by the data base administrator.

- Unlimited number of data relationships. Since pointers may be used to associate one file with another, the number of data relationships possible is unlimited. More data is therefore available to individual programs.

- Program development costs. A data base reduces the need for separate storage and retrieval procedures. Inconsistencies between individual files with redundant contents are reduced and program maintenance costs should decline with the elimination of some maintenance procedures.

Disadvantages

- Data integrity and data relationships. Since data items are normally stored only once within the data base, all users of a data item will access the same item. An incorrect data item will create an error for, and have a major impact on, all users. As the data base is enlarged, relationships each item has with the other items also increases. Therefore, those relationships must be defined accurately and each item in the relationship must always conform to its defined role.

- Problems for the auditors. Auditing a data base management system requires more sophisticated audit techniques. Traditional sequential file audit retrieval packages will not access data base management systems directly. Audit software for data base systems now is available but a learning curve will be in effect.

- Cost offsets with data bases. Increased costs in some areas may offset any cost reductions in program development. Increased costs may result from:
  - DBMS development.
  - Additional equipment.
  - Increased planning.
  - New and elaborate controls, requiring a more sophisticated programming staff.
  - Changes and adjustments to meet technological advances.
- Data security or privacy considerations. Files for all applications are generally mounted and available to application programs at all times. This differs from batch environment files which are dismounted and secured in a library when not in use. Risk exposure is often heightened by diffusion of file security responsibility. In batch systems, the files are the property of the various user departments. In a DBMS, data belongs to everyone so that no one user is solely responsible for its safety. Thus, control over these files rests with a central authority – such as the data base administrator – who determines who has access and the level of access capability, e.g., addition, deletion, modification, or read only. Because the data is available to many users from numerous terminals, it should be determined if levels of access to key fields are adequately controlled to maintain certain security and privacy of data.
- Destruction of files. Since massive volumes of data are difficult and costly to copy, data base systems when updated often destroy old files. This makes it more difficult to backup such files and requires periodically dumping files and saving transactions. Because of the expense and difficulty of such backup techniques, only critical data may be backed up; other less critical data is subject to destruction and loss.
- Software failure/slow responses. Because a data base system is so complex, there is a high probability of breakdowns. Precautions to avoid breakdowns can add another layer of overhead to the software which can slow response time.

Types of DBMS

The main types of data base structures are:

•Hierarchical – represented as a tree with the root at the top and data located where a tree would branch.

Figure 12.10
Sample Hierarchical Data Base

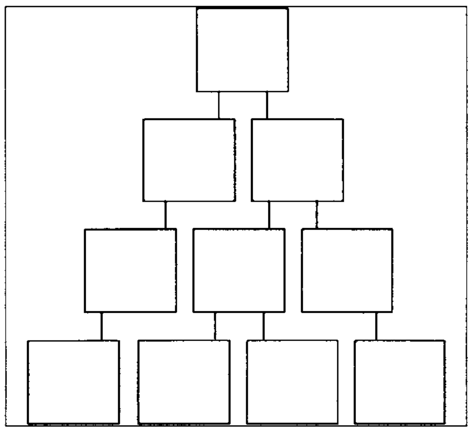
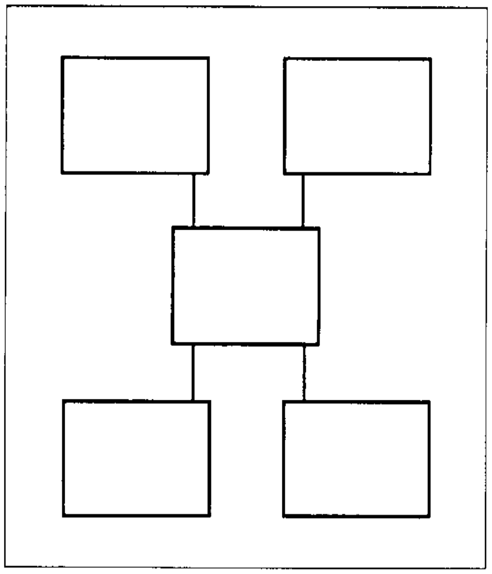


Figure 12.11
Sample Network Data Base



Similar to an organization chart, the lower data element is dependent on the one above it and a parent-child relationship can be envisioned as the data path. Each parent can have more than one child, but a child can have only one parent (Figure 12.10).

- Network – represented as links of parent/owner and child/member record types. Pointers can go in a forward or reverse direction and a child/member record can have more than one parent/owner. Within a network any data element can be connected to another one (Figure 12.11).

- Relational – represented as a table of columns and rows with columns corresponding to records and rows to fields. Paths or links are not pre-defined and no pointers are needed since relationships are defined at the data value level (Figure 12.12).

Refer to Figure 12.13 for examples of software for each data base structure. It is evident that relational data bases are becoming more prominent.

Data base systems traditionally have been centralized on one computer with data, programs and directories residing on a central processor. In a Distributed Data Base system, the data, program and directories can be spread among different processors and can be useful for organizations geographically dispersed or functionally diverse.

The essentials of data base security are:

- Data should be protected from fire, theft and other physical hazards.

Figure 12.12
Sample Relational Data Base

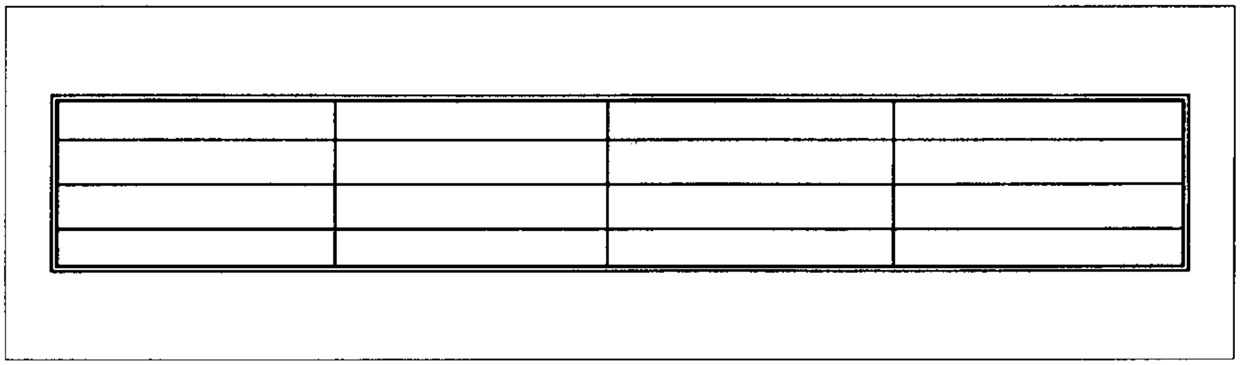


Figure 12.13
Example of DBMS Software

Hierarchical	Network	Relational
IMS (IBM)	CA-IDMS/DB	DB2 (IBM) (Computer Associates)
SYSTEM 2000 (INTEL)	TOTAL (Cincom)	SQL/DS (IBM)
	DMS 1100 (Unisys)	ORACLE (Oracle Corporation)
		RDB (Digital Equipment Corporation)
		CA-DATACOM/DB (Computer Associates)
		ADABAS (Software AG)
		MODEL 204 (CCA)
		INGRES (Ingres Corporation)
		INQUIRE (Infodata Systems)

- Data should be reconstructible to recover from destruction or loss (intentional or accidental).
- Data should be auditable for prompt detection of loss.
- The system should be tamper proof to prohibit programmers from bypassing controls.
- Users must be identified before being granted access.
- The system must be able to check that user actions are authorized.
- User actions should be monitored so that suspicious or unauthorized behavior can be investigated.

Threats present in all processing environments may be heightened by certain features of data base systems. These threats include:

- Deliberate internal threats.
- A trapdoor or window built into the system by a programmer. Provisions must be made for flexibility in the operating software, particularly in a system as complex as data base management. Since this software must be programmed for change, it is vulnerable to unauthorized attempts to penetrate the system.
- Dumping portions of core storage to obtain sensitive information, codes, passwords, etc. (This could be a profitable mode of attack because of the exposure of all files, and the heavy reliance on access codes to control input to the system.) A countermeasure is to encrypt sensitive information so that it is meaningless if dumped directly from storage.
- Misuse of another person's authorized access code, password, etc.
- Deliberate external threats.
- Obtaining access codes or other restricted data by obtaining copies of hard-copy output from trash cans, dumpsters, etc.
- Obtaining access codes or other restricted data by wiretapping.
- Accidental threats.
- Faults in the data communication system leading to erroneous data in the data base.
- Hardware or software failures leading to a breakdown of a built-in security feature. For example, secret data – such as passwords – may be printed as part of a core dump during an abnormal termination or a recovery.

Program Control

Data base management systems must rely heavily on software controls (as well as terminal hardware and physical controls) to compensate for loss of library controls present in the normal file-based systems. Software controls are divided into:

- Authorization controls – Specify whether access will be granted to the system in general, and to individual programs, files, or records. Also specify the functions that can be performed, e.g., read, write, modify.

- Processing controls – Include edit checks, such as validity tests, reasonableness tests, range and limit tests, and produce a full audit trail.

Control Differences Between Data Base and File Systems

File systems may use many physical controls that are not as readily available in data base systems. Therefore, data base systems rely more heavily on control over access to the input devices and on internal software controls. These controls include:

- Controls over input:
  - Physical terminal locks.
  - Locks on communications link with computer.
  - Terminal identification code/procedure.
  - User identification code/procedure.
  - User security code or password.
- Internal software controls:
  - Program locks (specific users may be prohibited from accessing specific programs).
  - File locks (specific users may be prohibited from accessing specific files).
  - Record or field locks (specific records or files may be restricted from access by specific user).
  - Tests of result (the processing result may prohibit both further action and the transmission of the result back to the user).

Files that require a high degree of security can be maintained on disk packs separate from other data base files. They can be mounted at only certain times of the day when high-security jobs are scheduled to run. When dismounted, they can be protected by rigorous security measures.

If it is not practical to copy large volumes of data for backup, or if multiple generations of files are not produced by the system, it is still possible to provide backup. Destructively updated files should be dumped periodically with the related transactions for the period between dumps. The files may be recreated from the last dump. However, a large data base usually will not be copied in its entirety. Therefore, it should be divided into vital and nonvital records and files, with emphasis placed on copying the vital information.

Certain data may change so infrequently that it is rarely copied and can be easily updated from transaction records. Some systems keep duplicate copies of critical file data and update both copies. If one file becomes unavailable or damaged, the second copy is immediately available. This may be important for certain critical files in a real-time system. However, the technique provides no protection against program errors since both copies can be damaged by the faulty program. In this case, it will be necessary to reconstruct from earlier file copies.

Data Base Administrator – Duties

The data base administrator serves as arbitrator and mediator in the development, use, and maintenance of the data base. The administrator coordinates the activities of the organization as they impact on the data base.

Specific functions of the data base administrator include:

- Data definition:
  - Naming data item types.
  - Detailing data items in the data dictionary.
  - Combining data items into groups.
  - Establishing the logical relationships between groups.
  - Establishing retention periods.
  - Establishing standards for data records and

files used by programmers.

- Data base design:
  - Physical structuring of data on media.
  - Developing access methods.
  - Designing restart and recovery procedures.
  - Developing security measures and techniques for maintaining privacy.
- Data operations:
  - Investigating all data errors.
  - Supervising restart and recovery operations.
  - Supervising reorganizations of the data base.
  - Controlling and documenting changes to the data base and educating users.
- Security:
  - Investigating all known security violations;
  - Investigating suspicious activity noted on exception reports.
  - Establishing an authorization hierarchy, i.e., who is entitled to access what data.
  - Modifying security codes, passwords, etc., as required.
  - Monitoring compliance with security procedures.
  - Conducting periodic security audits to test compliance.

The data base administrator may have one or more assistants to handle detail work, or the overall function may be divided among several people of relatively equal authority. The latter concept is particularly sound since different talents and temperaments are needed for designing operations and overseeing their day-to-day performance.

An effective separation of the data base administrator's duties must exist. Although this person is the guardian of the installation's information, access to application-related data should not be allowed. Access to source listings and other application program documentation also should be controlled.

Data Dictionary/Director System (DD/DS)

A data dictionary is a central catalog containing source data definitions for an organization. It also includes:

- Information confirming the existence and availability of data.
- Precise data item formats, segments, and other data base design mechanisms.
- Computerized and non-computerized data, conventional files, and data bases (the current state of data dictionary development normally includes only data base information).
- Object data definitions plus physical storage locations and data access methods.

Although most DBMS have built-in directories that function separately from the DD/DS, a DBMS generally does not store source definitions and usually maintains object definitions that relate to the data base only. A DD/DS may contain the source location for both conventional files and the data base and maintain some object definitions that are not always found in a DBMS, i.e., security access control methods at the data item level. Therefore, it tends to supplement rather than duplicate the library features of a DBMS. Data dictionaries can be either active/-integrated or passive/non-integrated. Active DDs are updated automatically whenever a change is made in the DBMS whereas passive DDs are non-integrated and must be maintained separately.

Standards and procedures must ensure that changes to the DD/DS will be monitored, documented and communicated to the proper personnel. Changes must be:

- Authorized through the data base administrator.
- Approved by users.
- Amended in user documentation.
- Adequately tested for impact on usage and linkages.
- Reviewed to ensure that audit trails and controls are maintained.

Data Base Monitoring

An essential feature of a data base system should be adequate transaction logs or journals. Some of the items that may be recorded and possible uses of such entries appear in the Sample System Protection and Auditing Logs as seen in Figure 12.14. These logs should be detailed enough to facilitate recovery from minor hardware failures, software failures and incidents where data base files have been damaged or destroyed. They should include the application transactions and the administrative and operator messages accompanying the transmission during processing.

In addition, a log or journal provides an audit trail to trace the history of a transaction and investigate the

Figure 12.14
Sample System Protection and Auditing Logs

Key
✓ = Yes
P = Possibly
I = At specified intervals

Functions:	To provide an audit trail for an auditor to follow the history of a transaction	To permit recovery when it is found that a user has incorrectly updated or deleted a record	To investigate the causes when a record is found to be faulty	To assist recovery from massive file destruction	To assist in correcting the file when a program has been damaging data	To correct false information which has been sent to system users	To monitor procedural violations to highlight possible breaches of security	To assist in correct recovery from a system failure	To monitor the way the system is being used (as an aid to design)	To recover from the loss of a file-action journal
Transaction journal										
Incoming inquiry transaction						✓			✓	
Incoming update transaction	✓	✓	✓			✓		✓	✓	✓
Transaction type		P	P			✓	✓		P	
Transaction number	✓	✓	✓			✓		✓		✓
Originating terminal	✓	✓	✓			✓	✓	✓	✓	✓
Originating operator	✓	✓	✓			✓	✓		✓	✓
Time and date										
Response to inquiry transaction	I	I	I			I	✓	I	I	I
Response to update transaction						✓			✓	
Indication that response was received correctly	✓	✓	✓			✓			✓	
Procedural violations on input	✓	✓	✓			✓		✓		✓
Record of start and end of file reconstruction							✓			
Note of completion of update									✓	
									✓	
File-action journal										
Transaction number	✓	✓	✓		✓					
Time and date		I	I	✓	I					
Addresses of items updated	✓	✓	✓	✓	✓					
Contents of items before they are updated										
Contents of items after they are updated	✓	✓	✓	✓	✓			✓		
Note of completion of update										
List of programs used for update	✓	✓	✓	✓	✓			✓		
Full contents of any records created			✓		✓					
Full contents of any records deleted			✓		✓					
Details of any indices opened or closed			✓		✓			✓		
Procedural violations deleted during update or processing			✓	✓	✓		✓	✓		
Contents of times corrected before correction	✓		✓	✓				✓		
Contents of times corrected after correction	✓		✓	✓				✓		
Indication of start and end of correction run	✓		✓	✓				✓		

The above information may all appear on the same journal.

cause of an error. The log also can highlight system utilization data and procedural violations that may indicate security breaches.

Some of the more advanced DBMS provide such logging features automatically; in others they are optional or must be developed by the user. More sophisticated control features provide automatic shutdown of disabled components and automatic transfer of functions to properly operating components.

COMPUTER–AIDED SOFTWARE ENGINEERING (CASE)

A software development technology known as Computer-Aided Software Engineering (CASE) may be used in financial institutions with large systems and programming staffs. CASE is used to automate software development process and maintain software systems in order to reduce programming costs, increase productivity, and produce reliable systems that satisfy users' needs. CASE technology provides a structured approach to designing information systems that meet business needs, manage information and share programming resources.

CASE uses a variety of process-modeling tools to automate systems design in all phases of the systems development life cycle (SDLC), e.g., data flow diagrams, structure charts, structure diagrams, data model diagrams, work breakdown structures, etc. Other products or tools provide the capability to create working models of a system's data entry screens, as well as layout of reports, for prototype testing by user departments. Structured system documentation can be automatically created in both hard copy and electronic form throughout the design and testing process.

CASE tools may be used separately or in an integrated fashion to reduce redundancy and increase coordination between programming tasks. The use of CASE tools in the early or late stages of the SDLC process are frequently referred to as UPPER CASE tools or LOWER CASE tools as described below.

- UPPER CASE refers to tools used in the early phases of systems design that relate to the requirements, analysis and design portions of SDLC.

- LOWER CASE refers to tools used in the later phases of the SDLC cycle that relate to natural language programming, documentation, code generators and testing.

CASE technology is often viewed as a solution to reduce software development problems. However, if not used with care, CASE tools can increase rather than reduce the risks associated with software development. Senior management and MIS directors need to be aware of these risk areas and controls.

Some common risk areas associated with the use of CASE technology are:

- Highly structured systems development – Institutions may move too quickly from an unstructured development approach to a highly structured CASE environment. CASE technology can be a radical change in the way systems are developed and tested. Systems and programming staffs may not adopt the processes necessary to implement CASE technology. Replacement of the old methods or simply implementing a methodology where none existed before cannot be accomplished quickly. Full CASE technology is generally implemented over a period of time to ensure staff and user departments are adequately trained in CASE.

- Change controls – An institution's use of CASE tools may affect the integrity of change controls in the institution's SDLC. The CASE products must be installed and administered properly. Generated code must be controlled by a sound change control process to assure it does not bypass control features that would allow a mismatch of source and object code.

- Unclear purpose – Management may fail to identify and quantify the purpose of acquiring and using CASE tools.

- Lack of standardization – CASE tools are not standard among vendors. CASE tools from one vendor may not be able to link to CASE tools from another vendor in the various stages of the SDLC. New tools on the market offer some integration, but only if the complete set is purchased. Typically, links between CASE tools and existing organization databases, dictionaries, and programming tools are inadequate, if they exist at all.

- Loss of CASE repository files – If critical files are destroyed, lost, or corrupted, the ability to regenerate any software may be lost.
- Project Selection – The project selected for

proving the value of CASE tools can be critical to the long term acceptance of CASE technology. The initial project should not be one that is critical to the organization, has high visibility, and short deadlines since there is a high learning curve during the transition to CASE tools. Likewise, attempting to use CASE tools to rescue a project that is over budget and late can be a mistake.

- Change in development process – Implementation of CASE programming tools will fundamentally alter the organization's software development culture and processes. Traditional SDLC concepts and procedures will have to be changed to take advantage of CASE. CASE methodology is highly structured and generally requires more discipline than developers and programmers may have been used to.

- High initial cost – CASE tools and associated staff training can be costly. Costs may appear to be initially high when weighed against early results before personnel have learned to use the tools efficiently and attained expected results.

- Management commitment – CASE cannot be implemented halfheartedly or part time. It requires a clear and strong management commitment and direction for success.

- Volume of output – A significant amount of documentation is generated when using CASE technology. The resources required to keep up with successive generations of documentation in order to know which change is in effect may require more staff time than expected.

- Training – Learning how to effectively use the new CASE tools requires a significant commitment of resources. This can lead to providing limited or no training in the hope that self-instruction will be sufficient; however, this is rarely the case.

In order to reduce the above risks and ensure maximum benefits, management should consider the following controls. These controls should not be regarded as all-inclusive nor should they be viewed as applicable in every situation.

- Clearly identify purpose and criteria for CASE Tools – Unless there is a clearly defined reason for acquiring CASE technology and specific requirements which must be met, there is reduced likelihood that it will be used efficiently or effectively. Often a cost/benefit analysis will help to clarify purpose and goals. Selection criteria should be determined prior to vendor discussions.

- Integrated CASE tools – Since CASE standards are incomplete today, the best way to ensure a smooth and automated interface between various tools is to consider purchase of CASE tools for the various stages of the SDLC from the same vendor and insist on demonstrable compatibility. If only selected CASE tools are going to be used, vendors should be selected that offer a full range of tools to reduce the likelihood that future CASE tools will be incompatible. If CASE tools are acquired from separate vendors, or one at a time, without regard to future interface, the likelihood that they will be incompatible is extremely high, with the result that a major benefit of CASE will have been lost.

- Protect the CASE repository – The CASE repository database should be backed up regularly, i.e., daily, secured from unauthorized access, and have controls over successive versions of data related to updated or revised development. The database usually contains all of the design and program data in order to generate and update software being developed. If this database is lost, the system progress to date and the ability to continue development also will be lost.

- Project selection – Until staff has experience with CASE tools, projects which have a low priority and are not time sensitive should be selected. If time pressures warrant, previous development methods can be reverted to. Likewise, projects which have high visibility run the same risk because of the pressure to meet deadlines. When CASE tools are not implemented successfully, benefits will not be realized, and acceptance will not be achieved.

- Determine systems development methodology changes – In addition to evaluating the changes in the development process required by the use of CASE tools, the flexibility of the existing policies and procedures to make necessary changes should be evaluated. Ample time should be allowed for changes to take place and expected benefits to be realized. It takes time for staff to learn to use the CASE tools effectively and efficiently.

- Quantify the increased productivity of CASE tools – The current productivity rate should be measured and compared with the productivity rates of CASE tools under consideration. The return on investment should be calculated to indicate if CASE tools should be acquired.

- Promote involvement throughout the organization – Once management decides to implement CASE technology tools, this commitment must be communicated and shared throughout the organization.

  - Senior management commitment and support of CASE technology are essential for success. Management must clearly communicate their intent to implement CASE tools and be willing to accept a long-term payback rather than quick-fix results.

  - Programmers and systems designers using the CASE technology should understand what effect CASE tools will have on their work. They are directly affected and their level of acceptance (or lack of it) in learning new concepts can determine the success of CASE projects.

  - The internal audit department (IAD) should be involved throughout the CASE implementation project. In the early stages, IAD can evaluate the proposed changes to modify the SDLC processes. IAD also can evaluate how well CASE was implemented and suggest improvements. CASE tools often generate general documentation which can be used as part of any audit. Repository reports can give the auditor an understanding of the systems development process.

  - The users probably will see parts of the proposed system or an early prototype sooner than under the traditional SDLC development process. They should be prepared for that early involvement and be ready to actively participate in testing and refining early models.

- Volume of output – Output from CASE tools should be retained on electronic media rather than hardcopy wherever possible. Strict procedures should be developed to control versions of each generated system or subsystem. Only those documents that are essential for current development or historical purposes should be retained and the rest discarded. Documentation stored on electronic media should be backed up on a regular basis.

- Training – CASE tools cannot be learned effectively from manuals or do-it-yourself efforts. Professional training is essential for mastering the tools and moving quickly up the learning curve. The cost of trial-and-error development can be much higher with accompanying delays.